



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение высшего  
образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

---

Институт информационных технологий (ИТ)  
Кафедра математического обеспечения и стандартизации информационных  
технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №8\_1**  
**по дисциплине**  
**«Структуры и алгоритмы обработки данных»**

Тема. Кодирование и сжатие данных методами без потерь

Выполнил студент группы ИКБО-42-23

Голев С.С.

Принял ассистент

Муравьёва Е.А.

Москва 2024

## ОГЛАВЛЕНИЕ

|  |    |
|--|----|
| ЦЕЛЬ РАБОТЫ.....                       | 3  |
| УСЛОВИЯ ЗАДАЧ.....                     | 4  |
| 1.ОТЧЁТ ПО ЗАДАНИЮ 1 .....             | 5  |
| 2. ОТЧЁТ ПО ЗАДАНИЮ 2 .....            | 9  |
| 2.1. Код используемый в программе..... | 9  |
| 2.3. Результаты тестирования .....     | 12 |
| 4.ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ .....         | 13 |

## **ЦЕЛЬ РАБОТЫ**

Получение практических навыков и знаний по выполнению сжатия данных рассматриваемыми методами.

## УСЛОВИЯ ЗАДАЧ

### *Задание 1:*

Исследование алгоритмов сжатия на примерах.

| LZ77                  | LZ78                  | методам Шеннона–Фано   |
|-----------------------|-----------------------|--|
| 010110110110100010001 | sarsalsarsanlasanl 33 | Мой котёнок очень странный, Он не хочет есть сметану, К молоку не прикасался И от рыбки отказался. |

### *Задание 2:*

Реализовать и отладить программу:

1. Разработать алгоритм и реализовать программу сжатия текста алгоритмом Шеннона – Фано. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на текстовом файле. Определить процент сжатия.
2. Применить алгоритм Хаффмана для архивации данных текстового файла. Выполнить практическую оценку сложности алгоритма Хаффмана. Провести архивацию этого же файла любым архиватором. Сравнить коэффициенты сжатия разработанного алгоритма и архиватора.

## 1. ОТЧЁТ ПО ЗАДАНИЮ 1

### Алгоритм 1 (RLE)

*Описание:* с помощью алгоритма мы проходим по всей строке, сперва мы ставим число – количество повторяющихся символов, а после уже сам символ.

*Пример:*

|             |                      |
|-------------|----------------------|
| Образец     | Выходная строка      |
| Коэффициент | 1K1o1э2ф1и1ц1и1e1н1т |

У данного алгоритма коэффициент сжатия этого текста составит: 1.9, в данном случае этот алгоритм не эффективен.

|                    |                  |
|--------------------|------------------|
| Образец            | Выходная строка  |
| aaabccaaaaabcbbbbc | 3a1b2c5a1b1c4b1c |

У данного алгоритма коэффициент сжатия этого текста составит: 0.8, в данном случае коэффициент хоть и меньше 1, но не на много.

### Алгоритм 2 (LZ77/LZ78)

*Описание:* данные методы относятся к “Словарным методам сжатия”, и в себе хранят в основном метки на смещения совпадений в строке.

**LZ77:**

Задача варианта: 010110110110100010001

| Строка                | совпадение | Последовательность |
|-----------------------|------------|--------------------|
| 010110110110100010001 | -          | (0,0,'0')          |
| 010110110110100010001 | -          | (0,0,'1')          |
| 010110110110100010001 | 01         | (2,2,'1')          |
| 010110110110100010001 | 011        | (3,3,'0')          |
| 010110110110100010001 | 1101       | (6,4,'0')          |
| 010110110110100010001 | 0          | (13,1,'0')         |
| 010110110110100010001 | 0          | (14,1,'0')         |
| 010110110110100010001 | 0          | (15,1,'0')         |
| 010110110110100010001 | 1000       | (4,4,'1')          |

**LZ78:**

Задача варианта: sarsalsarsanlasanl 33

| Код | Символ | Выходная строка    |
|-----|--------|--------------------|
| 1   | s      | 0s                 |
| 2   | a      | 0s0a               |
| 3   | r      | 0s0a0r             |
| 4   | sa     | 0s0a0r1a           |
| 5   | l      | 0s0a0r1a0l         |
| 6   | sar    | 0s0a0r1a0l4r       |
| 7   | san    | 0s0a0r1a0l4r4n     |
| 8   | la     | 0s0a0r1a0l4r4n5a   |
| 9   | sanl   | 0s0a0r1a0l4r4n5a7l |

**Алгоритм 3 (метод Шеннона-Фано)**

*Описание:* который строит префиксный код для символов исходя из их частоты встречаемости.

Пример: Мой котёнок очень странный, Он не хочет есть сметану, К молоку не прикасался И от рыбки отказался.

Посчитаем частоту символов:

| Символ | Частота | Символ | Частота | Символ | Частота |
|--------|---------|--------|---------|--------|---------|
| ‘ ’    | 15      | ё      | 1       | Ы      | 1       |
| о      | 10      | ч      | 1       |        |         |
| е      | 9       | р      | 1       |        |         |
| а      | 8       | ы      | 1       |        |         |
| н      | 7       | О      | 1       |        |         |
| т      | 7       | И      | 1       |        |         |
| А      | 6       | .      | 1       |        |         |
| К      | 6       | м      | 1       |        |         |
| С      | 4       | л      | 1       |        |         |
| ,      | 1       | у      | 1       |        |         |
| М      | 1       | и      | 1       |        |         |
| й      | 1       | п      | 1       |        |         |

Таблица формирования кода:

```
' ' -> 000
'a' -> 0101
'.' -> 111001
'o' -> 0010
'e' -> 1000
'н' -> 0011
'к' -> 011
'т' -> 0100
'с' -> 1001
'р' -> 10100
'л' -> 10101
'й' -> 101100
'ч' -> 101101
'ь' -> 10111
'ы' -> 110000
',' -> 11001
```

```
'м' -> 110001
'у' -> 110100
'и' -> 110101
'я' -> 11011
'М' -> 111000
'ё' -> 111010
'О' -> 1110110
'х' -> 1110111
'К' -> 111100
'п' -> 111101
'И' -> 111110
'6' -> 1111110
'з' -> 1111111
```

Коэффициент сжатия будет равен: **0.54**

Изначально символов **98**, но каждый символ вести 8 бит

В закодированном варианте **430** символ, но каждый вести 1 бит

#### **Алгоритм 4 (метод Хоффмана)**

*Описание:* который минимизирует среднюю длину кодирования за счет создания оптимального префиксного кода для символов. В отличие от метода Шеннона-Фано, алгоритм Хаффмана гарантирует построение минимальной длины кода.

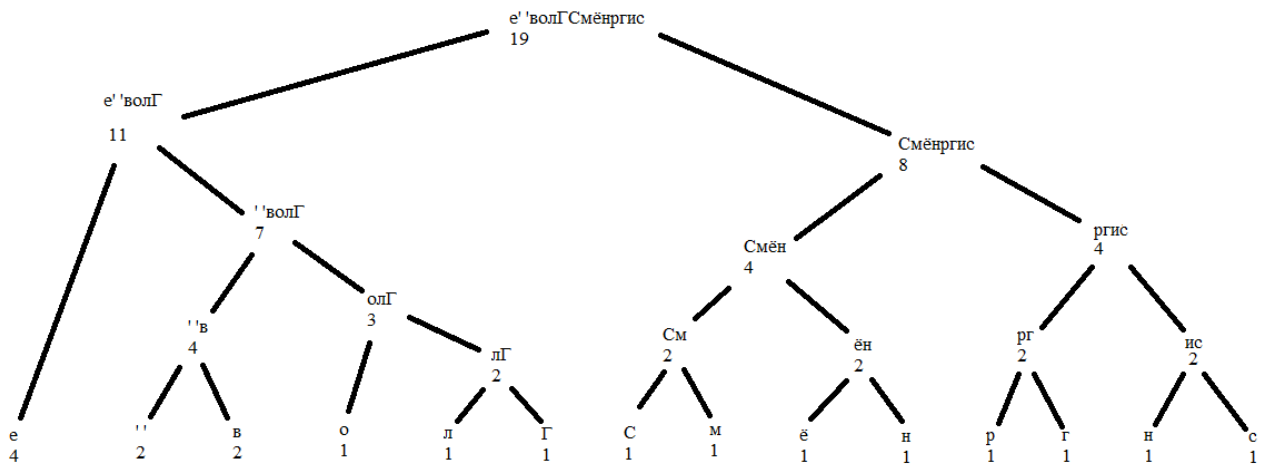
Пример: Голев Семён Сергеевич

Посчитаем частоту символов:

| Символ | Частота | Символ | Частота |
|--------|---------|--------|---------|
| е      | 4       | н      | 1       |
| ‘ ’    | 2       | р      | 1       |
| в      | 2       | г      | 1       |
| о      | 1       | и      | 1       |
| л      | 1       | с      | 1       |
| Г      | 1       |        |         |
| С      | 1       |        |         |
| м      | 1       |        |         |
| ё      | 1       |        |         |

Эти символы мы убираем в стек, сначала часто встречающиеся, потом редко встречающиеся. Далее достаём два верхних символа и создаём новый узел, у которого частота, это сумма частот этих символов.

Бинарное дерево:



Рассчитаем коэффициент сжатия: 0.45

Длина исходной строки 21\*8 бит.

Длина выходной строки 76 бит



## 2. ОТЧЁТ ПО ЗАДАНИЮ 2

Реализуем методы Шеннона-Фано и Хоффмана на языке C++

### 2.1. Код используемый в программе

Опишем функции, используемые в программе для решения задач.

```
std::string start_shannon(std::string& input)
{
    std::unordered_map<char, int> frequency_map;
    for (char c : input)
        frequency_map[c]++;

    std::vector<Symbol> symbols;
    for (const auto& pair : frequency_map)
        symbols.emplace_back(pair.first, pair.second);

    sort(symbols.begin(), symbols.end(), compare);
    std::unordered_map<char, std::string> codes;

    shannon_fano_encode(symbols, codes);
    return encode_string(input, codes);
}
```

*Рисунок 2.1 – Начальная функция для метода Шеннона-Фано*

```

void shannon_fano_encode(std::vector<Symbol>& symbols, std::unordered_map<char, std::string>&
codes, std::string prefix = "") {
    if (symbols.size() == 1)
    {
        codes[symbols[0].character] = prefix;
        return;
    }

    int total_frequency = 0;
    for (const auto& s : symbols)
        total_frequency += s.frequency;

    int cumulative_frequency = 0;
    int split_index = 0;

    for (int i = 0; i < symbols.size(); i++)
    {
        cumulative_frequency += symbols[i].frequency;
        if (cumulative_frequency >= total_frequency / 2)
        {
            split_index = i;
            break;
        }
    }

    std::vector<Symbol> symbolsL(symbols.begin(), symbols.begin() + split_index + 1);
    std::vector<Symbol> symbolsR(symbols.begin() + split_index + 1, symbols.end());
    shannon_fano_encode(symbolsL, codes, prefix + "0");
    shannon_fano_encode(symbolsR, codes, prefix + "1");
}

```

*Рисунок 2.2 – Рекурсивная функция построения дерева*

```

std::string encode_string(const std::string& input, const std::unordered_map<char, std::string>& codes) {
    std::string encoded = "";
    for (char c : input)
        encoded += codes.at(c);
    return encoded;
}

```

*Рисунок 2.3 – Функции кодирования строки на основе дерева*

```

std::string huffman_encode(const std::string& input)
{
    std::unordered_map<char, int> frequency;
    for (char c : input)
        frequency[c]++;

    std::priority_queue<Node*, std::vector<Node*>, Compare> minHeap;

    for (const auto& pair : frequency)
        minHeap.push(new Node(pair.first, pair.second));

    while (minHeap.size() > 1)
    {
        Node* left = minHeap.top();
        minHeap.pop();
        Node* right = minHeap.top();
        minHeap.pop();

        Node* newNode = new Node("\0", left->frequency + right->frequency);
        newNode->left = left;
        newNode->right = right;

        minHeap.push(newNode);
    }

    Node* root = minHeap.top();

    std::unordered_map<char, std::string> codes;
    generate_codes(root, "", codes);

    std::string encodedString = "";
    for (char c : input)
        encodedString += codes[c];

    return encodedString;
}

```

*Рисунок 2.4 – Начальная функция для метода Шеннона*

```

void generate_codes(Node* root, const std::string& code, std::unordered_map<char, std::string>& codes)
{
    if (!root) return;

    if (!root->left && !root->right)
        codes[root->character] = code;

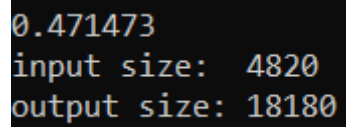
    generate_codes(root->left, code + "0", codes);
    generate_codes(root->right, code + "1", codes);
}

```

*Рисунок 2.5 – Функция формирования дерева*

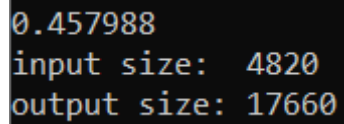
## 2.3. Результаты тестирования

Протестируем функции, используемые во втором задании.



```
0.471473  
input size: 4820  
output size: 18180
```

*Рисунок 2.6 – Тестирование реализации метода Шеннона-Фано*



```
0.457988  
input size: 4820  
output size: 17660
```

*Рисунок 2.7 – Тестирование реализации метода Хоффмана*

#### **4.ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ**

1. Лекции по Структуры и алгоритмы обработки данных / Рысин М. Л. Москва, МИРЭА — Российский технологический университет.
2. Материалы по дисциплине Структуры и алгоритмы обработки данных / Скворцова Л. А. Москва, МИРЭА — Российский технологический университет.